View metadata, citation and similar papers at core.ac.uk
brought to you by CORE
provided by UGD Academic Repository

Analysis of an Algorithm for finding Minimal Cut Set for Undirected Network

Marija Mihova[1] and Natasha Maksimova[2],

[1] Ss. Cyril an Methodius University, Faculty of Natural Sciencies and Mathematics, 1000 Skopje, Macedonia
[2]University "Goce Delčev, "Faculty of informatics, 2000 Shtip

Abstract. In this paper we propose an algorithm for obtaining all minimal cut sets for a given two-terminal network. The algorithm works on undirected networks without matter whether they are coherent or not. The difference between this algorithm and the other proposed algorithms is in the fact that there are not received candidates for minimal cut set that are not minimal cut sets. A large part of the paper proves the correctness of the algorithm and analyzes its complexity.

Keywords: Network, minimal cut set, cut set, undirected network, connected network, graph.

1 Introduction

One of the most complex problems in system reliability analysis is finding minimal path and cut sets. Minimal cut sets are mostly used for high reliability systems because this approach received minor rounding errors. A problem for developing algorithms that will give these sets is one of the frequently analyzed problems. But usually there are regarded directed and acyclic networks [1, 2]. Here we regard undirected network.

From the other side, in most of the proposed algorithms that solves this problem, there are obtained candidates for minimal cut sets that are not minimal. These sets must be eliminated by some additional procedure that involves comparison between all pairs of candidates for minimal cut set. This is an expensive procedure, without difference is there are a lot of such candidates or not. The algorithm proposed here gives minimal cut sets only. In fact we propose a technique for determination whether some cut set is a minimal cut set. Moreover we give a proof that the proposed algorithm gives all minimal cut sets. Additionally, we analyze the complexity of the algorithm.

2 General

Let we have a two-terminal undirected network $G(V, E)$, where V is the set of nodes, and E is a set of links. Let s be the source node and t be the sink node. The cut set is defined as a set of links, such that if there no flow through these links, then there no flow from the source to the sink. The cut set C is a minimal cut set if there is not another cut set C' such that $C' \subset C$. For an undirected network the following proposition is clear.

Proposition 1.1 Let $G(V, E)$ be an undirected connected network with source node s and sink node t. Then C is a cut set if and only if by removing the links from C, the graph $G(V, E)$ is divided into two subgraphs $G_1(V_1, E_1)$ and $G_2(V_2, E_2)$ such that $V_1 \cap V_2 = \varnothing$, $V_1 \cup V_2 = V$, the source $s \in V_1$ and the sink $t \in V_2$.

The following proposition specifies the minimal cut sets. The proof of it is given in [3].

Proposition 1.2 Let $G(V, E)$ be an undirected connected network with source node s and sink node t. If the graph $G(V, E)$ is divided into two connected subgraphs $G_1(V_1, E_1)$ and $G_2(V_2, E_2)$ such that $V_1 \cap V_2 = \varnothing$, $V_1 \cup V_2 = V$, the source $s \in V_1$ and the sink $t \in V_2$, then $C = E/(E_1 \cup E_2)$ is a minimal cut set.

Example 1.1 For the two-terminal network with source s and sink t on the Fig.1, the minimal cut set $C = \{\{s,c\}, \{c,b\}, \{b,e\}, \{e,d\}, \{d,t\}\}$ separated the network in to two subgraphs $G_1(\{s, a, b, d\}, \{\{s, a\}, \{s, b\}, \{a, d\}, \{b, d\}, \{a, b\}\})$ and $G_2(\{c, e, f, t\}, \{\{c, e\}, \{c, f\}, \{e, f\}, \{e, t\}, \{f, t\}\})$

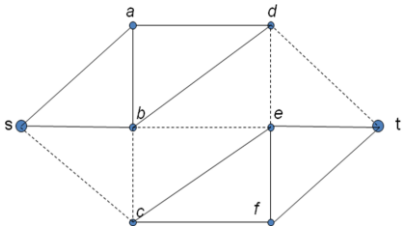

Fig. 1. The two-terminal network with source s and sink t. The set $C = \{\{s,c\}, \{c,b\}, \{b,e\}, \{e,d\}, \{d,t\}\}$is a minimal cut set for this network. The graph is separated in to $G_1(\{s, a, b, d\}, \{\{s, a\}, \{s, b\}, \{a, d\}, \{b, d\}, \{a, b\}\})$ and $G_2(\{c, e, f, t\}, \{\{c, e\}, \{c, f\}, \{e, f\}, \{e, t\}, \{f, t\}\})$

3 Description of the algorithm

In this section we will explain the work of the proposed algorithm. From the Proposition 1.2 follows that by deleting the edges from some minimal cut set C, the graph will be separate into exactly two connected components, such that, s belongs in the one of them and t in the other, Fig. 1 So, the idea of the algorithm is to construct all those components.

In order to avoid obtaining the same connected graphs more than once, we define ordering of the nodes. For this ordering we use BFS search starting from s and define a function

P: $V \rightarrow \mathbf{N}$, $P(a)=i$ if a is the i-th node in the BFS search started from s (1)

So, we have that if $P(a) < P(b)$ then the shortest path from s to a is longer or equal then the shorter path from s to b.

In each step of the while loop we use an element $\{B_1, B_2, B_3, T\}$. Let us see the meaning of each element from this list. B_1 is one minimal cut set for the given graph. Corresponding graph G_1 is $G_1(B_3, \{\{u, v\}|\{u, v\} \in B_3\})$ and B_2 is a set of all nodes v from V/B_3 that need to be added in B_3 in one of the next steps. T is a tree rooted in t that connects all nodes that are not in B_3. This tree is kept as a set of nodes, list of decedents of each node call `DescadentList` and the parent function π.

At the beginning we set the initial values of the sets B_1, B_2 and B_3. B_1 is a minimal cut set, consist of all links from s, B_3 is initialize on $\{s\}$, and B_2 is initialize on the set of all nodes connected by edge with s. The initialize tree T is a BFS tree of the graph $G'(V/B_3, E/B_1)$ rooted at t. It is clear that the minimal cut set B_1 divide the graph G into two graphs $G_1(\{s\}, \varnothing)$ and $G_2(\{V/\{s\}, E/B_1)$.

```
1 Program MINIMAL_CUT_SET (G, s, t)
  Output{CutSet (the set of all minimal cut sets)}

  Construct the function P using BFS started from s;
5 B1={{s, a} | {s, a} ∈ E};
  B2={a | {s, a} ∈ E}/{t};
  B3={s};
  T ={DescadentList, π} of the graph G'(V/B3,E/B1) by BFS;
  A = {{B1, B2, B3, T}};
10 MinCutSet={{{s, a} | {s, a} ∈ E}};
     while A≠∅ do
       Take an element B={B1, B2, B3, T} ∈ A and remove it from A.
       for each element b ∈ B2 do
         if first(CONECTION(E, B1, b, T))==True then
15         B3 = B3∪{b}
           B2 ={x|x∈ B2 ∧ P(b) < P(x)} ∪ {a | {a, b} ∉ B1 ∧ (P(b) < P(x) ∨
                                          ¬(∃{x, a}, {x, a}∈B1)}
           B1 = (B1\ {{a, b}|{a, b}∈B1})∪{{a, b}|{a, b} ∉B1}
```

```
                MinCutSet = MinCutSet ∪ {B1}
20              A = A∪{ {B1, B2, B3 , Last(CONECTION)}}
            else if MIN(CONNECTION[[2]]>b then
                B4 =CONECTION[[2]]
                B3 = B3∪{b} ∪B4
                B2  ={x|x∈ B2 ∧ P(b) < P(x)} ∪ {a | {a, b} ∉ B1 ∧ a ∉ B4 ∧
25                      (P(b) < P(x) ∨ ¬(∃{x, a}, {x, a}∈B1)}
                B1 = (B1\ {{a, b}|{a, b}∈B1})∪{{a, b}|{a, b} ∉B1∧ a ∉B4}
                MinCutSet = MinCutSet ∪ {B1}
                A = A∪{ {B1, B2, B3 , Last(CONECTION)}}
        Print (MinCutSet)
        end{program}
```

In each iteration of the while loop we take an element $\{B_1, B_2, B_3, T\}$ from A. The cut set B_1 divides G into two connected graphs $G_1(B_2, E_2)$ and $G_2(V/B_2, E/(E_2 \cup B_1))$. Then in each iteration of the **for** cycle we take one node from B_2 and check whether his removal from G_2 other nodes remain connected. This can be done by BFS starting from t. But we do this with the procedure `CONNECTION`, which is chipper. If the rest of G_2 is connected, in lines 15-20 we get another member in A and another minimal cut set.

If the rest of G_2 is not connected we do additional checks. Since we have a connected graph, they are connected by some node from $V \backslash V_T$, so V_T and $V \backslash V_T$ separate G into two connected components. Because we want no repetition of the same combination of vertices in B_3, we use strategy to add nodes having larger value of P. But it is not always possible to add nodes in increasing order, so when we receive a set of nodes that are not connected with t such that they have P value greater than $P(b)$, they are added to B_3 (22-28).

The procedure CONNECTION is called from the graph G, the tree T and one node b from T. The output of the procedure is an answer FAULT, when there are nodes in T that, after removing b, are not connected with the sink t and all of them have bigger value of P then $P(b)$. In opposite case, the answer is TRUE, together with a new tree rooted at t and list of unconnected nodes. At the beginning, the procedure CONNECTION (in lines 2 – 8) checks out whether b is a leaf in T. If it is true, than the tree obtained from T by removing b connects remain nodes of T. When b is not a leaf, we have more work. In lines 12-15 each descendant of b is color in red and removed from V_T. Then for each red element it is checked whether there is a link between it and some node from V_T (nodes for which we know that they are connected with the sink). If it is connected, it is colored in gray and added to the tree (d is put as a parent of c and c is put into DescadentList of d, lines 20-21). Now if there not gray nodes, it is clear that nodes in V_T are not connected, otherwise, we are not sure yet. So we use BFS, started from the gray nodes and add to the tree all gray nodes to which we arrive. If you still have red nodes, there are nodes that are not in V_T and we mark *isconect* as True. In opposite, we mark *isconect* as False.

```
1  Procedure CONNECTION (G, T=(V_T, DescedentList,π), b)
     Output{{True, T=(DescedentList,π)};

       If DescedentList(b)==∅ then            // if b is leaf
5        isconect = True
         remove b from the DescadentList(π(b))
         π(b)=NILL
       else
         Gray=∅
10       M=Red=DescedentList(b)
         VT=VT\M
         while M != ∅ do
                  c =First(M);
                  M=(M\{c})∪ DescedentList(c)
                  Red=Red∪ DescedentList(c);
15                VT=VT\ DescedentList(c)
          for all c ∈ Red do
                if there is d such that {c, d} ∈ E and d ∈ V_T
                        then
                          Gray=Gray ∪ {c};
20                        π(c)=d
                          DescedentList(d)= DescedentList(d) ∪ {c}
          while Gray != ∅ do
                f=First(Gray);
25              Gray=Gray/{f};
                VT=VT∪{f};
                DescedentList(f)={u|{u, f} ∈ V and  u ∈Red}
                Gray= Gray ∪ DescedentList(f)
                Red=Red/ DescedentList(f)
                for all u∈ DescedentList(f) do π(u)= f
30        if Red != ∅ then isconnect = False else isconnect = True
        if isconnect=False then output={False, Red, T=(VT, DescedentList,π)) }
          else output={True, T=(VT, DescedentList,π))
    print ( output )
    end{procedure}
```

As a illustration of the algorithm, we give following example.

Example 3.1. Let us consider how the algorithm works on the network given on Fig 2, with source node 1 and sink node 7. The tree rooted at 7 is shown on the Fig. 2 b.

At the beginning A={{B_1={{1, 2}, {1, 3}}, B_2={2, 3}, B_3={1}, T={{7,{2, 3}}, {2, {5, 4}}, {3, {6}}, {4, ∅}, {5, ∅}, {6, ∅}}}.

By the first step, when the procedure CONECTION is call for the first element 2, it is illustrates how the algorithm works with irrelevant links. 2 is not a leaf, so the nodes 4 and 5 are put in Red, and move from T. 4 is connected by 6, so it is color in gray and it is connects by the tree through the node 6. 5 is not connected with some

node from T, and after leaving the procedure CONECTION, Red={5}. New tree is {{7,{3}}, {3, {6}}, {6, {4}}, {4, ∅}} Now, min{5}=5>2, so we obtain the minimal cut set {{1, 3}, {2, 4}, {2, 7}}. In fact the element

{{{1, 3}, {2, 4}, {2, 7}}, {3, 4}, {1, 2, 5},{{{7,{3}}, {3, {6}}, {6, {4}}, {4, ∅}}} (2)

is added to A.

Next we call CONECTION for the element 3, which is a leaf, so the node 6 is put in Red, and move from T. But 6 is connect by 4 and Red becomes ∅. The element

{{{1, 2}, {3, 6}, {3, 7}}, {6}, {1, 3}, {{{7,{2}}, {2, {4, 5}}, {4, {6}}, {6, ∅}, {5, ∅}}} (3)

is added to A. This is an illustration of the case when the node *b* is not a leaf, but all other vertices are connected with the sink.

By the next step we also illustrate one of the characteristics cases. (2) is taken, and CONNECTION is called for 3. The nodes 6 and 4 are added into Red. When we remove 3, these nodes are not connected with the sink. But min{4, 5}=4>3, so 4 and 6 are added in B_3. So as a minimal cut set we obtain {{2, 7}, {3, 7}} and as a B_2, ∅. So from this element we do not obtain another cut set.

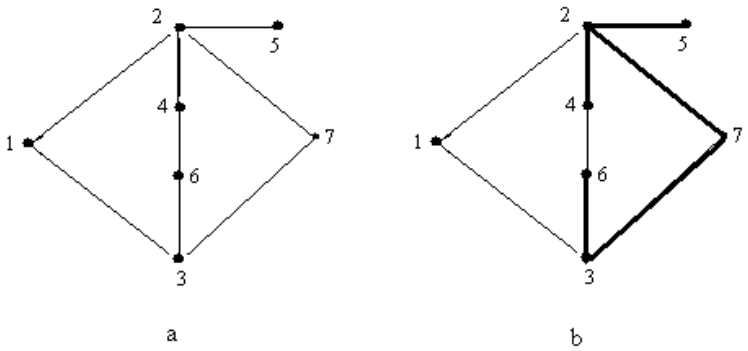

Fig.2. Two-terminal network with source node 1 and sink node 7.

Next step is a characteristics case when we add a leaf in B_3. In fact, CONNECTION is called for 4 which is a leaf in (2), so immediately we add

{{{1, 3}, {6, 4}, {2, 7}}, {6}, {1, 2, 4, 5},{{{7,{3}}, {3, {6}}, {6, ∅}}} (4)

into A.

We are finished with (2) and take (3). We have only one element in B_2, 6, which is a leaf and the cut set we obtained from here is {{1, 2}, {2, 7}, {6, 4}}. Node 4 is connected by node 6, and it is note found as a vertex in the set of links {{1, 2}, {3, 6}, {2, 7}}. So we add 4 in B_2 and put in A the element

{{{1, 2}, {4, 6}, {3, 7}}, {4}, {1, 3, 6}, {{{7, {2}}, {2, {4, 5}}, {4, ∅},{5, ∅}}} (5)

Now we take (4) and add the node 6 into B_3. B_2 becomes empty, since 6 is connect by 4 which is in B_3, and 3 which is a node in B_1. Tne minimal cut set obtained from here is {{1, 3}, {6, 3}, {2, 7}}. Similarly from (5) we obtain {{1, 2}, {4, 2}, {3, 7}} as a minimal cut set and the algorithm ends.

4 Analysis of the algorithm

This section will prove that the proposed algorithm for finding the minimal cut sets for a connected undirected network works correctly.

Proposition 2.1 Let MINIMAL_CUT_SET is terminate on the graph $G(V, E)$. After each iteration of the while loop, the graph $G_1(B_3, \{\{u, v\} | u, v \in B_3\})$ is a connected graph, such that $s \in B_3$.

Proof It is clear that $G_1(\{s\}, \emptyset)$ is a connected graph. In each iteration of the loop we take an element of A={B1, B2, B3, T} and add one new elements in B_3. Suppose that for all integer smaller or equal to k, if $|B_3| < k$, then $G_1(B_3, \{\{u, v\} | u, v \in B_3\})$ is connected. Let B'_3 is obtained from some B_3, such that $|B_3| < k$, by adding new set of nodes $Red \cup \{v\}$. It is clear that the subgraph of G consisting of these nodes is connected. Moreover, since G is connected graph, this subgraph is connected with some node $u \in B_3$. So B'_3 is also connected and $s \in B'_3 \subset B_3$.

Proposition 2.2 Let CONNECT is called from the connected graph G, the tree T of nodes from G, rooted in t and a node $b \neq t$ from T. Let $G_2(\{u | \text{ is in } T\}/\{b\}, \{\{u, v\} | u, v \in \{u | \text{ is in } T\}/\{b\})$. Then CONNECT gives False when G_2 is not connected, and True, together with a tree that connects the nodes from G_2, when G_2 is connected graph.

Proof First suppose that G_2 is not connected. Then there is a node u such that there is no link from t to u in G_2. We claim that all paths in $G'_2(\{u | u \text{ is in } T\}, \{\{u, v\} | u, v \in \{u | u \text{ is in } T\})$ pass through b (if it is not true, there is a path in G_2 from t to u). So, u must be a descendant of b in T and u is colored red in the line 14 and it is deleting from V_T in line 15. Since u is not connected with some node from V_T, u remains red after line 29. Now we claim that there is not a path from some gray node v, to u. If it is true, then there is a path $t \rightarrow a \rightarrow u \rightarrow v$, $\{a, u\} \in E$, $a \in V_T$ such that b is not on that path which is a contradiction. So, in line 30 the set Red$\neq\emptyset$ and CONNECT provides an answer False.

Now suppose that G_2 is connected. We need to proof that after the termination of the procedure CONNECT, $Red = \emptyset$. Since CONNECT is call when b is not a leaf, after line 16, $Red \neq \emptyset$. Let $u \in Red$. Since G_2 is connect, there is a path from t to u in G_2 do not passes through b. Suppose that the first node which lies on the path from t to u and it is a descendant of b in T is a. a will be added in to new tree in the lines 19-21. u will be added into new tree in lines 23-29, since this part of the algorithm is a part of BFS.

Theorem 2.1 Program MINIMAL_CUT_SET is consists of all minimal cut sets of a given graph $G(V, E)$.

Proof From Proposition 2.1 and Proposition 2.2 follows that each element we put into *CutSet* divide the graph G into two connected components, such that s is in one of them and t is in the other. So each element in *CutSet* is a minimal cut set.

It remains to prove that all minimal cut sets will be added to *CutSet*. This will be proved by induction in respect to the number of elements in B_3.

It is clear that the only set B_3 with one element is $\{s\}$. Suppose that all minimal cut sets that separate V into B_3 and V/ B_3, such that $|B_3|\leq k$, are added into *CutSet*. Let C divides G into two connected graphs $G_1(B_3, \{\{x, y\}| x, y \in B_3\})$ and $G_2(V/B_3, \{\{x, y\}| x, y \notin B_3\})$ and let $b \in B_3$ such that $P(b)=\max\{P(x)|x\in B_3$ and there is a link $\{x, y\}\in E$ and $y\notin B_3\}$. Define $S_0=\{b\}$, $S_k=\{x\in B_3| P(x) \geq P(b)$ and $\{x, y\}\in E, y \in S_{k-1}\}$. It is clear that there is finite number of such sets, so let $S = \bigcup_k S_k$ and that S_1 is the set of all neighbors of b in G_1. We will regard two cases:

I case: By removing the nodes from S, $G'_1(B_3\backslash S, \{\{x, y\}| x, y \in B_3\backslash S\})$ is connected. This case will be divided into two subcases, when all neighbors of b in G_1 have greater value of P then b and when there is a neighbor x of b in G_1 such that P(x)<P(b).

Regard how we will obtain C when all neighbors of b in G_1 have greater value of P then b, i.e. the set of neighbors of b in G_1 is S_1. Let a $\in S_1$ is the node with the smallest value of P such that it is connect with some node from $B_3\backslash (S_1\cup \{b\})$. Take the smallest subgraph from G_1, such that b and all neighbors of b in $G_1\backslash\{a\}$ are not in them. This graph and the rest of the graph G, (which is connect, since all nodes in it are connected by b and b is connect with the rest of the graph) separate G into two connected components and all links that are not in it constitute an minimal cut set. From the inductive assumption this minimal cut set is obtained by the algorithm. Moreover, b is in the corresponding list B_2, (it is add by the second part of the formula of B_2). Now, if $S=\{b\}$, C is obtained in lines 15-19, in opposite, in lines 22-27.

In the opposite, the graph G'_1 and the rest of the G separate G into two connected components and b is added into the corresponding list B_2 by the first part of the formula for B_2. So again if $S=\{b\}$, C is obtained in lines 15-19, in opposite, in lines 22-27.

II case: By removing the nodes from S, G'1(B3\S, {{x, y}| x, y ∈ B3\S}) is not connected. Let $\hat{G}'(\hat{V}', \hat{E}')$ is the connected component of G'1 in which s belongs, and $\hat{G}'$ is the graph $\hat{G}(\hat{V}'\cup S, \{\{x, y\} | x, y \in \hat{V}'\cup S\})$. Since for the nodes in G' that have smallest value of P then b, the shorter path in G is shorter then the shorter path in G'. This path must pass true nodes that are not in G', so, such nodes are connected by some other nodes from G2 i.e. $\hat{G}$ and $\hat{G}_1(V /(\hat{V}'\cup S), \{\{x, y\} | x, y \notin \hat{V}'\cup S\})$ separate G into two connected components and from the inductive assumption the appropriate minimal cut set is obtained. The nodes with smaller values of P are added in B_2 by the second part of the union.

Look at the graph $\hat{G}_1$. Let b' is a node in $\hat{G}_1$ such that $P(b')=\max\{P(x)|x\in B_3$, $x \in B_3 \cap (\hat{V}'\cup S)^C$ and there is a link $\{x, y\}\in E$ and y$\notin B_3\}$. Define $S'_0=\{b\}$, $S'_k=\{ x \in B_3 \cap (\hat{V}'\cup S)^C | P(x)\geq P(b')$ and $\{x, y\}\in E, y \in S'_{k-1}\}$ and $S' = \bigcup_k S'_k$. There two cases again, by removing the nodes from S', the rest of the graph is connected and by removing the nodes from S', the rest of the graph is unconnected. These two cases are considered in the same way as cases I and II. This can be

repeated until we get connected graph. Since G has a finite number of nodes, the procedure will finish.

At the and, let us regard the complexity of the proposed algorithm. Each minimal cut set is obtained only once, but, as a set of nodes B_3, there obtained some combination that not lead to minimal cut set. The good think is that each set B_3 is obtained at most once. So, the worst case is when all subsets of $V/\{t\}$ are acquired as a set B_3. This is obtained for complete graph only, but in this case, each candidate for a minimal cut set is actually a minimal cut set. From this we have that for a graph with |V| nodes the procedure CONNECTION is call at most $2^{|V|-1}$ times.

In the algorithm proposed in [1] has the same complexity in finding candidates for minimal cut sets, but the candidates for minimal cut set that are not minimal are rejected by comparison with all other candidates. In that comparison it is determine whether there is a set in the list of candidates for minimal cut set which is a subset of the given set of links, and if so, it is rejected from the list. So, in order to reject all candidates from the list of N candidates, it is needed to make $\Theta(N^2)$ comparisons between sets.

In our algorithm we have a procedure for determining whether a given cut set is minimal. In this procedure we make BFS on part of the network. In most of this cases there are very few nodes on which we make BFS. In fact, in dense graphs, the procedure CONNECTION is called frequently, but in most of the cases, the node from which it is called is a leaf or has a small number of descendents, so the complexity of the procedure is constant. For example, for complete graph, the procedure CONNECTION is called only for leaves.

6 Conclusion

The detailed analysis of the proposed algorithm indicates that it correctly gives all the minimal cut vectors for a given two-terminal undirected network. It is characteristic that this algorithm works for undirected networks that are less discussed in the literature. Because the cycles of such networks are inevitable, by minor modification it can be modified to work for directed networks in which the cycles are allowed. Also, this algorithm avoids getting cut sets that are not minimal cut sets. This avoids the additional complexity of the program as a result of rejection of those sets.

References

1. A.Bethel A. Gebre ,Jose E. Ramirez-Marquez, ”Element Substitution Algorithm for General Two-terminal Network Reliability Analyses”, IIE Transactions, Volume 39, March 2007, 265 - 275

2. A. Marteli, “A Gaussian Elimination Algorithm for the Enumeration of Cut Sets in a Graph”, Journal of the Association for Computing Machinery, Vol 23, No 1, January 1976
3. M. Mihova, N. Maksimova, “Minimal Cut Sets for Transportation System”, CD of extended abstracts of the 7th International Conference for Informatics and Information Technology (CIIT 2010);